# Automatic Determination of Communication Topologies in Mobile Systems

Arnaud Venet

LIX, École Polytechnique, 91128 Palaiseau, France.
venet@lix.polytechnique.fr
http://lix.polytechnique.fr/~venet

**Abstract.** The interconnection structure of mobile systems is very difficult to predict, since communication between component agents may carry information which dynamically changes that structure. In this paper we design an automatic analysis for statically determining all potential links between the agents of a mobile system specified in the $\pi$-calculus. For this purpose, we use a nonstandard semantics of the $\pi$-calculus which allows us to describe precisely the linkage of agents. The analysis algorithm is then derived by abstract interpretation of this semantics.

**Keywords.** $\pi$-calculus, nonstandard semantics, abstract interpretation.

## 1 Introduction

We are interested in analyzing the evolution of the interconnection structure, or *communication topology*, in a mobile system of processes, abstracting away all computational aspects but communication. Therefore, we can restrict our study to the $\pi$-calculus [Mil91, MPW92], which is a widely accepted formalism for describing communication in mobile systems. Whereas the communication topology of systems written in CSP [Hoa85] or CCS [Mil89] can be directly extracted from the text of the specification, a semantic analysis is required in the $\pi$-calculus, because communication links may be dynamically created between agents. In the absence of automatic analysis tools this makes the design and debugging of mobile systems very difficult tasks (see [DPLT96] for a detailed case study). In this paper we propose a semantic analysis of the $\pi$-calculus based on Abstract Interpretation [CC77, CC92] for automatically inferring approximate but sound descriptions of communication topologies in mobile systems.

In a previous work [Ven96b] we have presented an analysis of the $\pi$-calculus which relies on a nonstandard concrete semantics. In that model recursively defined agents are identified by the sequence of replication unfoldings from which they stem, whereas the interconnection structure is given by an equivalence relation on the agent communication ports. That semantics is inspired of a representation of sharing in recursive data structures [Jon81] which has been applied to alias analysis [Deu92]. However, the equivalence relation does not capture an important piece of information for debugging and verification purposes: the instance

of the channel that establishes a link between two agents. In this paper we redesign our previous analysis in order to take this information into account, while still preserving a comparable level of accuracy. Surprisingly enough, whereas our original analysis was rather complicated, involving heavy operations like transitive closure of binary relations, the refined one is tremendously simpler and only requires very basic primitives.

The paper is organized as follows. In Sect. 2 we introduce our representation of mobile systems in the $\pi$-calculus. Section 3 describes the nonstandard semantics of mobile systems, which makes instances of recursively defined agents and channels explicit. The abstract interpretation gathering information on communication topologies is constructed in Sect. 4. In Sect. 5 we design a computable analysis which is able to infer accurate descriptions of unbounded and nonuniform communication topologies. Related work is discussed in Sect. 6.

## 2 Mobile Systems in the $\pi$-Calculus

We consider the asynchronous version of the polyadic $\pi$-calculus which was introduced by Turner [Tur95] as a semantic basis of the PICT programming language. This restricted version has simpler communication primitives and a more operational flavour than the full $\pi$-calculus, while still ensuring a high expressive power[1]. Let $\mathcal{N}$ be a countable set of channel names. The syntax of processes is given by the following grammar:

$$
\begin{array}{lll}
P ::= & c![x_1, \ldots, x_n] & \text{Message} \\
\mid & c?[x_1, \ldots, x_n].P & \text{Input guard} \\
\mid & *c?[x_1, \ldots, x_n].P & \text{Guarded replication} \\
\mid & (P \mid P) & \text{Parallel composition} \\
\mid & (\nu x)P & \text{Channel creation}
\end{array}
$$

where $c, x, x_1, \ldots, x_n$ are channel names. Input guard and channel creation act as *name binders*, i.e. in the process $c?[x_1, \ldots, x_n].P$ (resp. $(\nu x)P$) the occurrences of $x_1, \ldots, x_n$ (resp. $x$) in $P$ are considered bound. Usual rules about scoping, $\alpha$-conversion and substitution apply. We denote by $\mathrm{fn}(P)$ the set of *free names* of $P$, i.e. those names which are not in the scope of a binder.

Following the CHAM style [BB92], the standard semantics of the $\pi$-calculus is given by a *structural congruence* and a *reduction relation* on processes. The congruence relation "$\equiv$" satisfies the following rules:

(i) $P \equiv Q$ whenever $P$ and $Q$ are $\alpha$-equivalent.
(ii) $P \mid Q \equiv Q \mid P$.
(iii) $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$.
(iv) $(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$.
(v) $(\nu x)P \mid Q \equiv (\nu x)(P \mid Q)$ if $x \notin \mathrm{fn}(Q)$.

The reduction relation is defined in Fig. 1, where $P\{x_1/y_1, \ldots, x_n/y_n\}$ denotes the result of substituting every name $x_i$ for the name $y_i$ in $P$. This may involve $\alpha$-conversion to avoid capturing one of the $x_i$'s.

[1] We can encode the lazy $\lambda$-calculus for example [Bou92].

$$c![x_1,\ldots,x_n] \mid c?[y_1,\ldots,y_n].P \to P\{x_1/y_1,\ldots,x_n/y_n\}$$

$$c![x_1,\ldots,x_n] \mid *c?[y_1,\ldots,y_n].P \to P\{x_1/y_1,\ldots,x_n/y_n\} \mid *c?[y_1,\ldots,y_n].P$$

$$\frac{P \to P'}{(\nu x)P \to P'} \qquad \frac{P' \equiv P \quad P \to Q \quad Q \equiv Q'}{P' \to Q'} \qquad \frac{P \to P'}{P \mid Q \to P' \mid Q}$$

**Fig. 1.** Reduction relation in the standard semantics.

We now have to define what we mean by a "mobile system in the $\pi$-calculus". We cannot simply allow a mobile system to be described by any process $S$. In fact, we are unable to design the nonstandard semantics, and hence the analysis, if we do not require $S$ to be *closed*, i.e. $\mathrm{fn}(S) = \emptyset$. In other words, we must consider the system in a whole. In order to make the semantic constructions of the following sections simpler, we add further constraints to the structure of mobile systems, which are inspired of Milner's definition of *friendly systems* [Mil91]. We denote by $\mathbf{x}$ a tuple $(x_1,\ldots,x_n)$ of channel names. We say that a process is a *thread* if it is made of a message possibly preceded by some input guards: $c_1?[\mathbf{x}_1]\ldots c_n?[\mathbf{x}_n].c![\mathbf{x}]$. We call *resource* a replicated process of the following form:

$$*c?[\mathbf{x}].(\nu\mathbf{y})(T_1 \mid \cdots \mid T_n)$$

where all the $T_i$'s are threads. A mobile system $S$ is then defined as:

$$S \equiv (\nu\mathbf{c})(R_1 \mid \cdots \mid R_n \mid T_0)$$

where the $R_i$'s are resources and $T_0$ is a message, the *initial thread*, which originates all communications in the system. The names in $\mathbf{c}$ are called the *initial channels*. Therefore, all threads and channels created by a mobile system are fetched from its resources.

**Example 1.** We model a system $S$ which sets up a ring of communicating processes, where each component agent may only communicate with its left and right neighbours. The system generating a ring of arbitrary size is defined as follows:

$$S \equiv (\nu\mathrm{make})(\nu\mathrm{mon})(\nu \mathit{left}_0)(R_1 \mid R_2 \mid \mathrm{make}![\mathit{left}_0])$$

where

$$R_1 \equiv *\mathrm{make}?[\mathit{left}].(\nu \mathit{right})(\mathrm{mon}![\mathit{left}, \mathit{right}] \mid \mathrm{make}![\mathit{right}])$$

is the resource that adds a new component to the chain of processes and

$$R_2 \equiv *\mathrm{make}?[\mathit{left}].\mathrm{mon}![\mathit{left}, \mathit{left}_0]$$

is the resource that closes the ring. The name "mon" should be seen as a reference to a hidden resource (for example a C program) which monitors the behaviour

of a ring component[2]. For the sake of clarity, we denote by $\mathbf{c}$ the free names of all agents in the system at every stage of its evolution. Then, a ring with four components can be generated by $S$ in four steps as follows:

$$\begin{aligned}
S &\to (\nu\mathbf{c})(R_1 \mid R_2 \mid \text{mon}![\mathit{left}_0, \mathit{right}_1] \mid \text{make}![\mathit{right}_1]) \\
&\to (\nu\mathbf{c})(R_1 \mid R_2 \mid \text{mon}![\mathit{left}_0, \mathit{right}_1] \mid \text{mon}![\mathit{right}_1, \mathit{right}_2] \\
&\qquad \mid \text{make}![\mathit{right}_2]) \\
&\to (\nu\mathbf{c})(R_1 \mid R_2 \mid \text{mon}![\mathit{left}_0, \mathit{right}_1] \mid \text{mon}![\mathit{right}_1, \mathit{right}_2] \\
&\qquad \mid \text{mon}![\mathit{right}_2, \mathit{right}_3] \mid \text{make}![\mathit{right}_3]) \\
&\to (\nu\mathbf{c})(R_1 \mid R_2 \mid \text{mon}![\mathit{left}_0, \mathit{right}_1] \mid \text{mon}![\mathit{right}_1, \mathit{right}_2] \\
&\qquad \mid \text{mon}![\mathit{right}_2, \mathit{right}_3] \mid \text{mon}![\mathit{right}_3, \mathit{left}_0])
\end{aligned}$$

The $\mathit{right}_i$'s represent the successive instances of the channel *right* created at each request to $R_1$. □

As illustrated by the above example, the configuration of a mobile system at any stage of its evolution has the following particular form:

$$(\nu\mathbf{c})(\underbrace{R_1 \mid \cdots \mid R_m}_{\text{Resources}} \mid \underbrace{T_1 \mid \cdots \mid T_n}_{\text{Threads}})$$

where $R_1, \ldots, R_m$ are the resources originally present in the system. Intuitively, every thread $T_i$ or channel $c_i$ present in the configuration could be unambiguously identified with the instant of its creation in the history of computations. Unfortunately, this precious information is not captured by the standard semantics. The process of $\alpha$-conversion in particular, which is inherent to the definition of the semantics, destroys the identity of channels. The purpose of the next section is to introduce a refined semantics of the $\pi$-calculus which restores this information.

## 3 Nonstandard Semantics of Mobile Systems

Let $S$ be a mobile system described in the $\pi$-calculus. In order to identify the threads and channels created by the system, we must be able to locate the syntactic components of $S$ from which they stem. This is the role of the following notations. We denote by $\mathrm{R}(S)$ the number of resources in $S$. We assume that every resource is assigned a unique number $r$ in $\{1, \ldots, \mathrm{R}(S)\}$. For any such $r$, we denote by $S_r$ the corresponding resource in $S$ and by $\mathrm{T}(r)$ the number of threads spawned by the resource. We similarly assign a unique number $t$ in $\{1, \ldots, \mathrm{T}(r)\}$ to every thread in $S_r$ and we denote by $S_{r,t}$ this thread, which has the following form:

$$S_{r,t} = c_1?[\mathbf{x}_1] \ldots c_{\mathrm{A}(r,t)-1}?[\mathbf{x}_{\mathrm{A}(r,t)-1}].c_{\mathrm{A}(r,t)}![\mathbf{x}_{\mathrm{A}(r,t)}]$$

[2] Recall that we only take communications into account, abstracting away all other computational aspects like the details of the monitoring procedure here.

where $\mathrm{A}(r,t)$ is the number of input/output actions in $S_{r,t}$. Note that $\mathrm{A}(r,t)$ is always nonzero because a thread contains at least one message. For $1 \leqslant n \leqslant \mathrm{A}(r,t)$, we denote by $\mathrm{act}(r,t,n)$ the $n$-th input/output action of $S_{r,t}$, and by $S_{r,t} \mathbin{@} n$ the subterm $c_n?[\mathbf{x}_n] \ldots c_{\mathrm{A}(r,t)}![\mathbf{x}_{\mathrm{A}(r,t)}]$ of $S_{r,t}$ starting at the $n$-th input/output action $\mathrm{act}(r,t,n)$. By convention, the initial thread is assigned the resource number 0. Finally, given a resource number $r$ such that

$$S_r \equiv *c?[\mathbf{x}].(\nu y_1) \ldots (\nu y_n)(T_1 \mid \cdots \mid T_{\mathrm{T}(r)})$$

we put $\mathrm{guard}(r) = c?[\mathbf{x}]$ and $\mathrm{C}(r) = \{y_1, \ldots, y_n\}$.

**Example 2.** We consider the system of Example 1 which contains two resources $R_1$ and $R_2$. We put $S_1 = R_1$ and $S_2 = R_2$. Thus we have:

- $\mathrm{guard}(1) = \mathrm{make}?[\mathit{left}], \mathrm{C}(1) = \{\mathit{right}\}, \mathrm{T}(1) = 2, S_{1,1} = \mathrm{mon}![\mathit{left}, \mathit{right}]$, $S_{1,2} = \mathrm{make}![\mathit{right}], \mathrm{A}(1,1) = \mathrm{A}(1,2) = 1$.
- $\mathrm{guard}(2) = \mathrm{make}?[\mathit{left}], \mathrm{C}(2) = \emptyset, \mathrm{T}(2) = 1, S_{2,1} = \mathrm{mon}![\mathit{left}, \mathit{left}_0], \mathrm{A}(2,1) = 1$.

The initial thread is $S_{0,1} = \mathrm{make}![\mathit{left}_0]$. □

A configuration of $S$ in the nonstandard semantics is a finite set of *thread instances*. We do not need to represent resources since they are statically given and accessible by all the threads in any state of the system. A thread instance is a tuple $\langle r, t, n, \mathit{id}, E \rangle$ where $1 \leqslant r \leqslant \mathrm{R}(S)$, $1 \leqslant t \leqslant \mathrm{T}(r)$, $1 \leqslant n \leqslant \mathrm{A}(r,t)$, $\mathit{id}$ is a *thread identifier* and $E$ is an *environment*. The thread identifier is the history of resource requests which led to the creation of the thread, starting from the initial one. A resource request being nothing more than a message to a replicated process, it can be identified with the thread that released this message. If we denote by

$$\mathrm{Thr}(S) = \{(r,t) \mid 1 \leqslant r \leqslant \mathrm{R}(S), 1 \leqslant t \leqslant \mathrm{T}(r)\}$$

the set of all threads originally present in the system, then $\mathit{id} \in \mathrm{Thr}(S)^*$. The empty sequence $\varepsilon$ is the identifier of the initial thread. The environment $E$ maps every free name $x \in \mathrm{fn}(S_{r,t} \mathbin{@} n)$ of the thread instance to a *channel instance*. A channel instance is a tuple $(r', y, \mathit{id}')$ where $1 \leqslant r' \leqslant \mathrm{R}(S)$, $y \in \mathrm{C}(r')$ and $\mathit{id}'$ is a *channel identifier*. Instances of initial channels are represented similarly except that they are assigned the resource number 0. The channel identifier is the history of resource requests that led to the creation of the instance of channel $y$ by resource $r'$. Therefore, channel identifiers and thread identifiers are represented identically, i.e. $\mathit{id}' \in \mathrm{Thr}(S)^*$. Similarly, the identifier of an initial channel is the empty sequence $\varepsilon$.

We assume that there is no overlapping of scopes in the mobile system, i.e. we forbid terms like $x?[y].y?[y].y![z]$ or $*c?[y,z].(\nu y)z![y]$. This assumption can always be satisfied by appropriate $\alpha$-conversion. The transition relation "$\triangleright$" of the nonstandard operational semantics is defined in Fig. 2 and Fig. 3. It should be clear that without the hypothesis on name scoping, the definition of the resulting environments is ambiguous. The two transition rules correspond to the

two kinds of operations that may arise in a mobile system: resource fetching and communication between threads. The initial configuration $C_0$ of the nonstandard semantics is given by $C_0 = \{\langle 0, 1, 1, \varepsilon, E_0\rangle\}$ where the environment $E_0$ maps any $z \in \mathrm{fn}(S_{0,1} \mathbin{@} 1)$ to the instance $(0, z, \varepsilon)$ of the corresponding initial channel.

**Example 3.** Using the notations of Example 2, a ring with four components is described in the nonstandard semantics as follows:

$$\left\{ \begin{array}{l} \left\langle 1,1,1,(0,1), \left\{ \begin{array}{l} \mathrm{mon} \mapsto (0,\mathrm{mon},\varepsilon), \\ \mathit{left} \mapsto (0,\mathit{left}_0,\varepsilon), \\ \mathit{right} \mapsto (1,\mathit{right},(0,1)) \end{array} \right\} \right\rangle, \\ \left\langle 1,1,1,(0,1).(1,2), \left\{ \begin{array}{l} \mathrm{mon} \mapsto (0,\mathrm{mon},\varepsilon), \\ \mathit{left} \mapsto (1,\mathit{right},(0,1)) \\ \mathit{right} \mapsto (1,\mathit{right},(0,1).(1,2)) \end{array} \right\} \right\rangle, \\ \left\langle 1,1,1,(0,1).(1,2)^2, \left\{ \begin{array}{l} \mathrm{mon} \mapsto (0,\mathrm{mon},\varepsilon), \\ \mathit{left} \mapsto (1,\mathit{right},(0,1).(1,2)) \\ \mathit{right} \mapsto (1,\mathit{right},(0,1).(1,2)^2) \end{array} \right\} \right\rangle, \\ \left\langle 2,1,1,(0,1).(1,2)^3, \left\{ \begin{array}{l} \mathrm{mon} \mapsto (0,\mathrm{mon},\varepsilon), \\ \mathit{left} \mapsto (1,\mathit{right},(0,1).(1,2)^2) \\ \mathit{left}_0 \mapsto (0,\mathit{left}_0,\varepsilon) \end{array} \right\} \right\rangle \end{array} \right\}$$

The generative process of the ring is now made entirely explicit thanks to the information carried by thread and channel identifiers (compare with the corresponding computation in the standard semantics given in example 1). □

Both semantics can be shown to be equivalent by defining the translation of a nonstandard configuration $C$ of $S$ to a term $\pi(C)$ of the $\pi$-calculus and by applying a *bisimulation* argument. We may assume without loss of generality that the $\pi$-terms we will generate are built upon the set of names $\mathcal{N}'$ defined as the disjoint union of $\mathcal{N}$ with the set of all channel instances $(r, y, id)$ that could be created by $S$. We identify an environment with a substitution over $\mathcal{N}'$. We denote by $P\sigma$ the result of applying a substitution $\sigma$ to a $\pi$-term $P$. Then, for any nonstandard configuration

$$C = \{\langle r_1, t_1, n_1, id_1, E_1\rangle, \ldots, \langle r_k, t_k, n_k, id_k, E_k\rangle\}$$

we define the translation $\pi(C)$ as follows:

$$\pi(C) = (\nu \mathbf{c})(S_1 E'_1 \mid \cdots \mid S_{\mathrm{R}(S)} E'_{\mathrm{R}(S)} \mid (S_{r_1,t_1} \mathbin{@} n_1)E_1 \mid \cdots \mid (S_{r_k,t_k} \mathbin{@} n_k)E_k)$$

where, for $1 \leqslant r \leqslant \mathrm{R}(S)$, $E'_r$ is the environment which maps any $z \in \mathrm{fn}(S_r)$ to $(0, z, \varepsilon)$. The channels in $\mathbf{c}$ are all those which have a free occurrence in some agent of the top-level parallel composition of $\pi(C)$.

**Theorem 4.** *If $C_0 \overset{*}{\triangleright} C$ and $C \triangleright C'$, then $\pi(C) \to \pi(C')$. If $C_0 \overset{*}{\triangleright} C$ and $\pi(C) \to P$, then there exists $C'$ such that $C \triangleright C'$ and $P \equiv \pi(C')$.*

If there are $\mu \in C$ and $1 \leqslant r' \leqslant \mathrm{R}(S)$ such that:

- $\mu = \langle r, t, n, id, E \rangle$
- $\mathrm{act}(r, t, n) = x![x_1, \ldots, x_k]$
- $\mathrm{guard}(r') = c?[y_1, \ldots, y_k]$
- $E(x) = (0, c, \varepsilon)$

then

$$C \triangleright (C - \{\mu\}) \cup \{\langle r', t', 1, id.(r,t), E_{t'} \rangle \mid 1 \leqslant t' \leqslant \mathrm{T}(r')\}$$

where, for all $1 \leqslant t' \leqslant \mathrm{T}(r')$ and $z \in \mathrm{fn}(S_{r',t'} \mathbin{@} 1)$

$$E_{t'}(z) = \begin{cases} E(x_i) & \text{if } z = y_i,\ 1 \leqslant i \leqslant k \\ (r', z, id.(r,t)) & \text{if } z \in \mathrm{C}(r') \\ (0, z, \varepsilon) & \text{otherwise, i.e. if } z \text{ is an initial channel} \end{cases}$$

**Fig. 2.** Resource fetching.

A nonstandard configuration describes the communication topology of a mobile system at a particular moment of its evolution in terms of the resources initially present in the system. Therefore, the nonstandard semantics is a good basis for deriving an analysis, since the resulting information can be used to determine the role of each *syntactic* component of the system in the evolution of its interconnection structure. Constructing a computable abstraction of this semantics is the purpose of the next section.

## 4 Abstract Interpretation of Mobile Systems

We denote by $\mathcal{C}$ the set of all possible nonstandard configurations for a system $S$. We are actually interested in the set $\mathcal{S} = \{C \in \mathcal{C} \mid C_0 \overset{*}{\triangleright} C\}$ of configurations of the system which are accessible from the initial one by a finite sequence of computations. This is the *collecting semantics* of $S$ [Cou81], which can be expressed as the least fixpoint of the $\cup$-complete endomorphism $\mathbb{F}$ on the complete lattice $(\wp(\mathcal{C}), \subseteq, \cup, \emptyset, \cap, \mathcal{C})$ defined as follows:

$$\mathbb{F}(X) = \{C_0\} \cup \{C \mid \exists C' \in X : C' \triangleright C\}$$

Following the methodology of Abstract Interpretation [CC77, CC92], we construct a lattice $(\mathcal{C}^\sharp, \sqsubseteq, \sqcup, \bot, \sqcap, \top)$, the *abstract domain*, allowing us to give finite descriptions of infinite sets of configurations. This domain is related to $\wp(\mathcal{C})$ via a monotone map $\gamma : (\mathcal{C}^\sharp, \sqsubseteq) \to (\wp(\mathcal{C}), \subseteq)$, the *concretization function*. Then we derive an abstract counterpart $\mathbb{F}^\sharp : \mathcal{C}^\sharp \to \mathcal{C}^\sharp$ of $\mathbb{F}$ which must be *sound* with respect to $\gamma$, that is: $\mathbb{F} \circ \gamma \subseteq \gamma \circ \mathbb{F}^\sharp$.

The abstract domain $\mathcal{C}^\sharp$ is based upon a global abstraction of all environments in a nonstandard configuration. We assume that we are provided

If there are $\mu, \rho \in C$ such that:

- $\mu = \langle r, t, n, id, E\rangle$
- $\rho = \langle r', t', n', id', E'\rangle$
- $\mathrm{act}(r, t, n) = x![x_1, \ldots, x_k]$
- $\mathrm{act}(r', t', n') = y?[y_1, \ldots, y_k]$
- $E(x) = E'(y)$

then

$$C \triangleright (C - \{\mu, \rho\}) \cup \{\langle r', t', n' + 1, id', E''\rangle\}$$

where, for all $z \in \mathrm{fn}(S_{r',t'} \mathbin{@} n' + 1)$

$$E''(z) = \begin{cases} E(x_i) & \text{if } z = y_i,\ 1 \leqslant i \leqslant k \\ E'(z) & \text{otherwise} \end{cases}$$

**Fig. 3.** Communication between threads.

with a lattice $(\mathrm{Id}_2^\sharp, \sqsubseteq_2, \sqcup_2, \bot_2, \sqcap_2, \top_2)$ and a monotone map $\gamma_2 : (\mathrm{Id}_2^\sharp, \sqsubseteq_2) \to (\wp(\mathrm{Thr}(S)^* \times \mathrm{Thr}(S)^*), \subseteq)$. This lattice is left as a parameter of our abstraction. It will be instantiated in the next section when we will set up an effective analysis. Let $\mathrm{Chan}(S)$ be the set $\{(r, t, n, x, r', y) \mid (r, t) \in \mathrm{Thr}(S), n \in \mathrm{A}(r, t), x \in \mathrm{fn}(S_{r,t} \mathbin{@} n), 1 \leqslant r' \leqslant \mathrm{R}(S), y \in \mathrm{C}(r')\}$ of all possible syntactic relations between a free name in a thread and a channel created by a resource of the system. The abstract domain $\mathcal{C}^\sharp$ is then defined as $\mathcal{C}^\sharp = \mathrm{Chan}(S) \to \mathrm{Id}_2^\sharp$, the lattice operations being the pointwise extensions of those in $\mathrm{Id}_2^\sharp$. Given an abstract configuration $C^\sharp$, $\gamma(C^\sharp)$ is the set of nonstandard configurations $C$ such that, for any $\langle r, t, n, id, E\rangle \in C$ and any $x \in \mathrm{fn}(S_{r,t} \mathbin{@} n)$, the following condition is satisfied:

$$E(x) = (r', y, id') \Rightarrow (id, id') \in \gamma_2(C^\sharp(r, t, n, x, r', y))$$

Monotonicity of $\gamma$ is readily checked. For the sake of readability, we will denote an abstract configuration $C^\sharp$ by its graph $\{\langle \varkappa_1 \mapsto id_1^\sharp\rangle, \ldots, \langle \varkappa_n \mapsto id_n^\sharp\rangle\}$, where the $\varkappa_i$'s are in $\mathrm{Chan}(S)$ and each $id_j^\sharp$ is in $\mathrm{Id}_2^\sharp$. We may safely omit to write pairs of the form $\langle \varkappa \mapsto \bot_2\rangle$.

The abstract semantics is given by a transition relation $\rightsquigarrow$ on abstract configurations. In the relation $C_1^\sharp \rightsquigarrow C_2^\sharp$, the configuration $C_2^\sharp$ represents the *modification* to the communication topology of $C_1^\sharp$ induced by an abstract computation. Therefore, the function $\mathbb{F}^\sharp$ is given by:

$$\mathbb{F}^\sharp(C^\sharp) = C_0^\sharp \sqcup C^\sharp \sqcup \bigsqcup \{\overline{C}^\sharp \mid C^\sharp \rightsquigarrow \overline{C}^\sharp\}$$

where $C_0^\sharp$ is the initial abstract configuration defined as $\{\langle 0, 1, 1, x, 0, x\rangle \mapsto \varepsilon_2 \mid x \in \mathrm{fn}(S_{0,1} \mathbin{@} 1)\}$, $\varepsilon_2$ being a distinguished element of $\mathrm{Id}_2^\sharp$ such that $(\varepsilon, \varepsilon) \in \gamma_2(\varepsilon_2)$. The transition relation $\rightsquigarrow$ is defined in Fig. 4 and Fig. 5 by using the

If there are $\langle (r,t,n,x,0,c) \mapsto id^\sharp \rangle \in C^\sharp$ and $1 \leqslant r' \leqslant \mathrm{R}(S)$ such that:

- $\mathrm{act}(r,t,n) = x![x_1,\ldots,x_k]$
- $\mathrm{guard}(r') = c?[y_1,\ldots,y_k]$
- $id^\sharp \neq \bot_2$

then

$$C^\sharp \rightsquigarrow \{\langle \varkappa \mapsto id^\sharp_{t',z,r'',w} \rangle \mid \varkappa = (r',t',1,z,r'',w), 1 \leqslant t' \leqslant \mathrm{T}(r')\}$$

where, for all $1 \leqslant t' \leqslant \mathrm{T}(r')$

$$id^\sharp_{t',z,r'',w} = \begin{cases} \mathbf{push}_{(r,t)}(C^\sharp(r,t,n,x_i,r'',w)) & \text{if } z = y_i, 1 \leqslant i \leqslant k \\ \bot_2 & \text{if } z \in \mathrm{C}(r') \text{ and } (r'',w) \neq (r',z) \\ \mathbf{dpush}_{(r,t)}(C^\sharp(r,t,n,x,0,c)) & \text{if } z \in \mathrm{C}(r') \text{ and } (r'',w) = (r',z) \\ \bot_2 & \text{if } z \text{ is initial and } (r'',w) \neq (0,z) \\ \mathbf{spush}(C^\sharp(r,t,n,x,0,c)) & \text{if } z \text{ is initial and } (r'',w) = (0,z) \end{cases}$$

**Fig. 4.** Abstract resource fetching.

following abstract primitives: a "push" operation $\mathbf{push}_\tau : \mathrm{Id}_2^\sharp \to \mathrm{Id}_2^\sharp$, a "double push" $\mathbf{dpush}_\tau : \mathrm{Id}_2^\sharp \to \mathrm{Id}_2^\sharp$ and a "single push" $\mathbf{spush}_\tau : \mathrm{Id}_2^\sharp \to \mathrm{Id}_2^\sharp$ defined for any $\tau \in \mathrm{Thr}(S)$, a "synchronization" operator $\mathbf{sync} : \mathrm{Id}_2^\sharp \times \mathrm{Id}_2^\sharp \to \mathrm{Id}_2^\sharp$ and a "swapping" operation $\mathbf{swap} : \mathrm{Id}_2^\sharp \to \mathrm{Id}_2^\sharp$. These operations depend on the choice of $\mathrm{Id}_2^\sharp$, however they must satisfy some soundness conditions:

- For any $\tau \in \mathrm{Thr}(S)$ and $id^\sharp \in \mathrm{Id}_2^\sharp$, $\{(id.\tau, id') \mid (id,id') \in \gamma_2(id^\sharp)\} \subseteq \gamma_2(\mathbf{push}_\tau(id^\sharp))$.
- For any $\tau \in \mathrm{Thr}(S)$ and $id^\sharp \in \mathrm{Id}_2^\sharp$, $\{(id.\tau, id.\tau) \mid (id,id') \in \gamma_2(id^\sharp)\} \subseteq \gamma_2(\mathbf{dpush}_\tau(id^\sharp))$.
- For any $\tau \in \mathrm{Thr}(S)$ and $id^\sharp \in \mathrm{Id}_2^\sharp$, $\{(id.\tau, \varepsilon) \mid (id,id') \in \gamma_2(id^\sharp)\} \subseteq \gamma_2(\mathbf{spush}_\tau(id^\sharp))$.
- For any $id^\sharp \in \mathrm{Id}_2^\sharp$, $\{(id',id) \mid (id,id') \in \gamma_2(id^\sharp)\} \subseteq \gamma_2(\mathbf{swap}(id^\sharp))$.
- For any $id_1^\sharp, id_2^\sharp \in \mathrm{Id}_2^\sharp$, $\{(id_1,id_2) \mid \exists id' \in \mathrm{Thr}(S)^* : (id_1,id') \in \gamma_2(id_1^\sharp) \wedge (id_2,id') \in \gamma_2(id_2^\sharp)\} \subseteq \gamma_2(\mathbf{sync}(id_1^\sharp, id_2^\sharp))$.

Intuitively, the $\mathbf{push}_\tau$ operation concatenates $\tau$ to the first component of every pair of identifiers. The $\mathbf{dpush}_\tau$ and $\mathbf{spush}_\tau$ operations act similarly, except that $\mathbf{dpush}_\tau$ duplicates the first component into the second one and $\mathbf{spush}_\tau$ sets the second component to $\varepsilon$. The **swap** operation permutes the components of every pair of identifiers. The **sync** operation extracts pairs of identifiers corresponding to redexes, i.e. pairs of agents linked to the same instance of a channel.

**Proposition 5.** *If $C \in \gamma(C^\sharp)$ and $C \triangleright C'$, then there exists $\overline{C}^\sharp$ in $C^\sharp$, such that $C^\sharp \rightsquigarrow \overline{C}^\sharp$ and $C' \in \gamma(C^\sharp \sqcup \overline{C}^\sharp)$.*

If there are $\langle \varkappa_1 \mapsto id_1^\sharp \rangle, \langle \varkappa_2 \mapsto id_2^\sharp \rangle \in C^\sharp$ such that:

- $\varkappa_1 = (r, t, n, x, r'', u)$
- $\varkappa_2 = (r', t', n', y, r'', u)$
- $\mathrm{act}(r, t, n) = x![x_1, \ldots, x_k]$
- $\mathrm{act}(r', t', n') = y?[y_1, \ldots, y_k]$
- $id_s^\sharp = \mathbf{sync}(id_1^\sharp, id_2^\sharp) \neq \bot_2$

then

$$C^\sharp \rightsquigarrow \{\langle \varkappa \mapsto id^\sharp_{z,r''',w} \rangle \mid \varkappa = (r', t', n'+1, z, r''', w)\}$$

where, if we put $\overline{id}^\sharp_{i,r''',w} = C^\sharp(r, n, t, x_i, r''', w)$ for $1 \leqslant i \leqslant k$

$$id^\sharp_{z,r''',w} = \begin{cases} \mathbf{swap}(\mathbf{sync}(\mathbf{swap}(\overline{id}^\sharp_{i,r''',w}), \mathbf{swap}(id_s^\sharp))) & \text{if } z = y_i,\ 1 \leqslant i \leqslant k \\ C^\sharp(r', t', n', z, r''', w) & \text{otherwise} \end{cases}$$

**Fig. 5.** Abstract communication between threads.

The soundness of $\mathbb{F}^\sharp$ is then a simple consequence of the previous result. It is remarkable that the soundness of the whole semantics depends only on very simple conditions over some primitive operations. This means that we only need to construct the domain $\mathrm{Id}_2^\sharp$ and instantiate those operations to obtain a computable and sound abstract semantics. This is the goal that we will achieve in the next section.

## 5 Design of a Computable Analysis

The collecting semantics $\mathcal{S}$ is the least fixpoint of $\mathbb{F}$. Therefore, by Kleene's theorem, it is the limit of the following increasing iteration sequence:

$$\begin{cases} \mathcal{S}_0 & = \emptyset \\ \mathcal{S}_{n+1} & = \mathbb{F}(\mathcal{S}_n) \end{cases}$$

Following [CC77, Cou81] we will compute a sound approximation $\mathcal{S}^\sharp$ of $\mathcal{S}$ by mimicking this iteration, using $\mathbb{F}^\sharp$ instead of $\mathbb{F}$. Since the resulting computation may not terminate, we use a *widening* operator to enforce convergence in finitely many steps. A widening operator $\nabla : \mathcal{C}^\sharp \times \mathcal{C}^\sharp \to \mathcal{C}^\sharp$ must satisfy the following conditions:

- For any $C_1^\sharp, C_2^\sharp \in \mathcal{C}^\sharp$, $C_1^\sharp \sqcup C_2^\sharp \sqsubseteq C_1^\sharp \nabla C_2^\sharp$.
- For any sequence $(C_n^\sharp)_{n \geqslant 0}$, the sequence $(\overline{C}_n^\sharp)_{n \geqslant 0}$ defined as:

$$\begin{cases} \overline{C}_0^\sharp & = C_0^\sharp \\ \overline{C}_{n+1}^\sharp & = \overline{C}_n^\sharp \nabla C_{n+1}^\sharp \end{cases}$$

is ultimately stationary.

Note that we can construct a widening on $\mathcal{C}^\sharp$ from an existing widening $\nabla_2$ on $\mathrm{Id}_2^\sharp$ by pointwise application of $\nabla_2$. We define the approximate iteration sequence $(\mathcal{S}^\sharp{}_n)_{n\geqslant 0}$ as follows:

$$\begin{cases} \mathcal{S}^\sharp{}_0 & = \bot \\ \mathcal{S}^\sharp{}_{n+1} = \mathcal{S}^\sharp{}_n \nabla \mathbb{F}^\sharp(\mathcal{S}^\sharp{}_n) & \text{if} \quad \neg(\mathbb{F}^\sharp(\mathcal{S}^\sharp{}_n) \sqsubseteq \mathcal{S}^\sharp{}_n) \\ \mathcal{S}^\sharp{}_{n+1} = \mathcal{S}^\sharp{}_n & \text{if} \quad \mathbb{F}^\sharp(\mathcal{S}^\sharp{}_n) \sqsubseteq \mathcal{S}^\sharp{}_n \end{cases}$$

Convergence is ensured by the following result:

**Theorem 6 [Cou81].** *The sequence $(\mathcal{S}^\sharp{}_n)_{n\geqslant 0}$ is ultimately stationary and its limit $\mathcal{S}^\sharp$ satisfies $\mathcal{S} \subseteq \gamma(\mathcal{S}^\sharp)$. Moreover, if $N \geqslant 0$ is such that $\mathcal{S}^\sharp{}_{N+1} = \mathcal{S}^\sharp{}_N$, then for all $n \geqslant N$, $\mathcal{S}^\sharp{}_n = \mathcal{S}^\sharp{}_N$.*

This provides us with an algorithm for automatically computing a sound approximation of $\mathcal{S}$. It now remains to instantiate the domain $\mathrm{Id}_2^\sharp$ and the associated abstract primitives. We will design two abstractions of $\wp(\mathrm{Thr}(S)^* \times \mathrm{Thr}(S)^*)$, each one capturing a particular kind of information.

Our first abstraction captures sequencing information and is based upon an approximation of thread and channel identifiers by *regular languages*. Let $(\mathrm{Reg}, \subseteq, \cup, \emptyset, \cap, \mathrm{Thr}(S)^*)$ be the lattice of regular languages over the alphabet $\mathrm{Thr}(S)$ ordered by set inclusion. We define $\mathrm{Id}_{\mathrm{reg}}^\sharp$ as the product lattice $\mathrm{Reg} \times \mathrm{Reg}$. The concretization $\gamma_{\mathrm{reg}}$ is given by $\gamma_{\mathrm{reg}}(L_1, L_2) = L_1 \times L_2$. The associated abstract primitives are defined as follows:

- $\mathbf{push}_\tau^{\mathrm{reg}}(L_1, L_2) = (L_1.\tau, L_2)$
- $\mathbf{dpush}_\tau^{\mathrm{reg}}(L_1, L_2) = (L_1.\tau, L_1.\tau)$
- $\mathbf{spush}_\tau^{\mathrm{reg}}(L_1, L_2) = (L_1.\tau, \varepsilon)$
- $\mathbf{swap}^{\mathrm{reg}}(L_1, L_2) = (L_2, L_1)$
- $\mathbf{sync}^{\mathrm{reg}}((L_1, L_2), (L_1', L_2')) = \begin{cases} (L_1, L_1') & \text{if } L_2 \cap L_2' \neq \emptyset \\ (\emptyset, \emptyset) & \text{otherwise} \end{cases}$

The soundness conditions are easily checked. The element $\varepsilon_2^{\mathrm{reg}}$ is given by $(\varepsilon, \varepsilon)$. Since $\mathrm{Id}_{\mathrm{reg}}^\sharp$ may have infinite strictly increasing chains, we must define a widening operator $\nabla_2^{\mathrm{reg}}$. It is sufficient to construct a widening $\nabla_{\mathrm{reg}}$ on Reg and to apply it componentwise to elements of $\mathrm{Id}_{\mathrm{reg}}^\sharp$. A simple choice for $L_1 \nabla_{\mathrm{reg}} L_2$ consists of quotienting the minimal automaton of $L_1 \cup L_2$ such that any letter of the alphabet may occur at most once in the automaton. The resulting automaton is minimal, and there are finitely many such automata, which ensures the stabilization property.

The second approximation captures counting relations between the components of a tuple of thread or channel identifiers. This will allow us to give nonuniform descriptions of recursively defined communication topologies. Suppose that we are given an infinite set of variables $\mathcal{V}$. We assign two distinct variables $x_\tau$ and $y_\tau$ to each element $\tau$ of $\mathrm{Thr}(S)$. Now we consider a finite system $K$ of *linear equality constraints* over the variables $\mathcal{V}$ with coefficients in $\mathbb{Q}$. If we denote

by $|id|_\tau$ the number of occurrences of $\tau$ in the sequence $id$, the concretization $\gamma_{\text{num}}(K)$ of $K$ is the set of all pairs $(id, id')$ such that the following variable assignment:

$$\{x_\tau \mapsto |id|_\tau, y_\tau \mapsto |id'|_\tau \mid \tau \in \text{Thr}(S)\}$$

is a solution of $K$. The domain of finite systems of linear equality constraints over $\mathcal{V}$ ordered by inclusion of solution sets can be turned into a lattice $\text{Id}^\sharp_{\text{num}}$. This domain has been originally introduced by Karr [Kar76]. We refer the reader to the original paper for a detailed algorithmic description of lattice operations. We could use more sophisticated domains of computable numerical constraints such as *linear inequalities* [CH78] or *linear congruences* [Gra91], but the underlying algorithmics is much more involved. Nevertheless, giving a rigorous construction of the abstract primitives on $\text{Id}^\sharp_{\text{num}}$ would be still very technical. Therefore, for the sake of readability, we only outline the definition of these primitives:

- $\mathbf{push}^{\text{num}}_\tau(K)$ is the system of constraints $K$ in which we have replaced every occurrence of the variable $x_\tau$ by the expression $x_\tau - 1$.
- If $K$ is a system of linear equality constraints, we denote by $K_x$ the system in which we have removed all constraints involving a variable $y_\tau$. Then $\mathbf{dpush}^{\text{num}}_\tau(K)$ is the system $\mathbf{push}^{\text{num}}_\tau(K_x)$ with the additional constraints $x_{\tau'} = y_{\tau'}$, for any $\tau' \in \text{Thr}(S)$.
- Similarly, $\mathbf{spush}^{\text{num}}_\tau(K)$ is the system $\mathbf{push}^{\text{num}}_\tau(K_x)$ with the additional constraints $y_{\tau'} = 0$, for any $\tau' \in \text{Thr}(S)$.
- $\mathbf{swap}^{\text{num}}(K)$ is the system in which we have replaced each occurrence of $x_\tau$ by $y_\tau$ and vice-versa.
- Let $K_1$ and $K_2$ be two systems of linear equality constraints. For any $\tau \in \text{Thr}(S)$, let $x'_\tau$ and $y'_\tau$ be fresh variables of $\mathcal{V}$. Let $K'_2$ be the system $K_2$ in which we have substituted each occurrence of $x_\tau$ (resp. $y_\tau$) by $x'_\tau$ (resp. $y'_\tau$). We construct the system $K_{1,2}$ as the union of $K_1$ and $K'_2$ together with the additional constraints $y_\tau = y'_\tau$, for any $\tau \in \text{Thr}(S)$. Then, we define $\mathbf{sync}^{\text{num}}(K_1, K_2)$ as the system $K_{1,2}$ in which we have removed all constraints involving a variable $y_\tau$ or $y'_\tau$, each remaining variable $x'_\tau$ being renamed in $y_\tau$.

Note that a normalization pass (namely a Gauss reduction) has to be performed on the system after or during each of these operations. The element $\varepsilon_2^{\text{num}}$ is given by the system of constraints $\{x_\tau = 0, y_\tau = 0 \mid \tau \in \text{Thr}(S)\}$. Since we only consider systems defined over the finite set of variables $\{x_\tau, y_\tau \mid \tau \in \text{Thr}(S)\}$, we cannot have infinite strictly increasing chains [Kar76]. Therefore, we can use the join operation $\sqcup_{\text{num}}$ as a widening.

**Example 7.** We consider the product of domains $\text{Id}^\sharp_{\text{reg}}$ and $\text{Id}^\sharp_{\text{num}}$ and we run the analysis on the system of Example 1 with the notations of Example 2. For the sake of readability, at each step we only write the elements of the abstract configuration that differ from the previous iteration. Moreover, we do not figure trivial constraints of the form $x_\tau = 0$ whenever they can be deduced from the $\text{Id}^\sharp_{\text{reg}}$ component.

**First iteration.**

$$\left\{\begin{array}{ll}\langle 0,1,1,\mathrm{make},0,\mathrm{make}\rangle & \mapsto \langle(\varepsilon,\varepsilon),\top_{\mathrm{num}}\rangle,\\ \langle 0,1,1,\mathit{left}_0,0,\mathit{left}_0\rangle & \mapsto \langle(\varepsilon,\varepsilon),\top_{\mathrm{num}}\rangle\end{array}\right\}$$

**Second iteration.**

$$\left\{\begin{array}{ll}\langle 1,1,1,\mathrm{mon},0,\mathrm{mon}\rangle & \mapsto \langle((0,1),\varepsilon),x_{(0,1)}=1\rangle,\\ \langle 1,1,1,\mathit{left},0,\mathit{left}_0\rangle & \mapsto \langle((0,1),\varepsilon),x_{(0,1)}=1\rangle,\\ \langle 1,1,1,\mathit{right},1,\mathit{right}\rangle & \mapsto \langle((0,1),(0,1)),x_{(0,1)}=y_{(0,1)}=1\rangle,\\ \langle 1,2,1,\mathrm{make},0,\mathrm{make}\rangle & \mapsto \langle((0,1),\varepsilon),x_{(0,1)}=1\rangle,\\ \langle 1,2,1,\mathit{right},1,\mathit{right}\rangle & \mapsto \langle((0,1),(0,1)),x_{(0,1)}=y_{(0,1)}=1\rangle,\\ \langle 2,1,1,\mathrm{mon},0,\mathrm{mon}\rangle & \mapsto \langle((0,1),\varepsilon),x_{(0,1)}=1\rangle,\\ \langle 2,1,1,\mathit{left}_0,0,\mathit{left}_0\rangle & \mapsto \langle((0,1),\varepsilon),x_{(0,1)}=1\rangle,\\ \langle 2,1,1,\mathit{left},0,\mathit{left}_0\rangle & \mapsto \langle((0,1),\varepsilon),x_{(0,1)}=1\rangle\end{array}\right\}$$

**Third iteration.**

$$\left\{\begin{array}{ll}\langle 1,1,1,\mathrm{mon},0,\mathrm{mon}\rangle & \mapsto \langle((0,1).(\varepsilon+(1,2)),\varepsilon),x_{(0,1)}=1\rangle,\\ \langle 1,1,1,\mathit{left},0,\mathit{right}\rangle & \mapsto \left\langle((0,1).(1,2),(0,1)),\left\{\begin{array}{l}x_{(0,1)}=y_{(0,1)}=1\\ x_{(1,2)}=1\end{array}\right.\right\rangle,\\ \langle 1,1,1,\mathit{right},1,\mathit{right}\rangle & \mapsto \left\langle\begin{array}{l}((0,1).(\varepsilon+(1,2)),(0,1).(\varepsilon+(1,2))),\\ \left\{\begin{array}{l}x_{(0,1)}=y_{(0,1)}=1\\ x_{(1,2)}=y_{(1,2)}\end{array}\right.\end{array}\right\rangle,\\ \langle 1,2,1,\mathrm{make},0,\mathrm{make}\rangle & \mapsto \langle((0,1).(\varepsilon+(1,2)),\varepsilon),x_{(0,1)}=1\rangle,\\ \langle 1,2,1,\mathit{right},1,\mathit{right}\rangle & \mapsto \left\langle\begin{array}{l}((0,1).(\varepsilon+(1,2)),(0,1).(\varepsilon+(1,2))),\\ \left\{\begin{array}{l}x_{(0,1)}=y_{(0,1)}=1\\ x_{(1,2)}=y_{(1,2)}\end{array}\right.\end{array}\right\rangle,\\ \langle 2,1,1,\mathrm{mon},0,\mathrm{mon}\rangle & \mapsto \langle((0,1).(\varepsilon+(1,2)),\varepsilon),x_{(0,1)}=1\rangle,\\ \langle 2,1,1,\mathit{left}_0,0,\mathit{left}_0\rangle & \mapsto \langle((0,1).(\varepsilon+(1,2))),\varepsilon),x_{(0,1)}=1\rangle,\\ \langle 2,1,1,\mathit{left},0,\mathit{right}\rangle & \mapsto \left\langle((0,1).(1,2),(0,1)),\left\{\begin{array}{l}x_{(0,1)}=y_{(0,1)}=1\\ x_{(1,2)}=1\end{array}\right.\right\rangle\end{array}\right\}$$

**Fourth iteration.**

$$\left\{\begin{array}{ll}\langle 1,1,1,\mathrm{mon},0,\mathrm{mon}\rangle & \mapsto \langle((0,1).(1,2)^*,\varepsilon),x_{(0,1)}=1\rangle,\\ \langle 1,1,1,\mathit{left},0,\mathit{right}\rangle & \mapsto \left\langle\begin{array}{l}((0,1).(1,2)^*,(0,1).(\varepsilon+(1,2))),\\ \left\{\begin{array}{l}x_{(0,1)}=y_{(0,1)}=1\\ x_{(1,2)}=y_{(1,2)}+1\end{array}\right.\end{array}\right\rangle,\\ \langle 1,1,1,\mathit{right},1,\mathit{right}\rangle & \mapsto \left\langle\begin{array}{l}((0,1).(1,2)^*,(0,1).(1,2)^*),\\ \left\{\begin{array}{l}x_{(0,1)}=y_{(0,1)}=1\\ x_{(1,2)}=y_{(1,2)}\end{array}\right.\end{array}\right\rangle,\\ \langle 1,2,1,\mathrm{make},0,\mathrm{make}\rangle & \mapsto \langle((0,1).(1,2)^*,\varepsilon),x_{(0,1)}=1\rangle,\\ \langle 1,2,1,\mathit{right},1,\mathit{right}\rangle & \mapsto \left\langle\begin{array}{l}((0,1).(1,2)^*,(0,1).(1,2)^*),\\ \left\{\begin{array}{l}x_{(0,1)}=y_{(0,1)}=1\\ x_{(1,2)}=y_{(1,2)}\end{array}\right.\end{array}\right\rangle,\\ \langle 2,1,1,\mathrm{mon},0,\mathrm{mon}\rangle & \mapsto \langle((0,1).(1,2)^*,\varepsilon),x_{(0,1)}=1\rangle,\\ \langle 2,1,1,\mathit{left}_0,0,\mathit{left}_0\rangle & \mapsto \langle((0,1).(1,2)^*,\varepsilon),x_{(0,1)}=1\rangle,\\ \langle 2,1,1,\mathit{left},0,\mathit{right}\rangle & \mapsto \left\langle\begin{array}{l}((0,1).(1,2)^*,(0,1).(\varepsilon+(1,2))),\\ \left\{\begin{array}{l}x_{(0,1)}=y_{(0,1)}=1\\ x_{(1,2)}=y_{(1,2)}+1\end{array}\right.\end{array}\right\rangle\end{array}\right\}$$

**Fifth iteration.**

$$\left\{\begin{array}{l}\langle 1,1,1,\mathit{left},0,\mathit{right}\rangle \mapsto \left\langle \begin{array}{l}((0,1).(1,2)^*,(0,1).(1,2)^*),\\ \left\{\begin{array}{l} x_{(0,1)} = y_{(0,1)} = 1\\ x_{(1,2)} = y_{(1,2)} + 1\end{array}\right.\end{array}\right\rangle,\\ \langle 2,1,1,\mathit{left},0,\mathit{right}\rangle \mapsto \left\langle \begin{array}{l}((0,1).(1,2)^*,(0,1).(1,2)^*),\\ \left\{\begin{array}{l} x_{(0,1)} = y_{(0,1)} = 1\\ x_{(1,2)} = y_{(1,2)} + 1\end{array}\right.\end{array}\right\rangle\end{array}\right\}$$

At the sixth iteration step we find the same configuration. Therefore, following Theorem 6, we know that the limit has been reached. Putting all previous computations together, we obtain:

$$\mathcal{S}^\sharp = \left\{\begin{array}{ll}\langle 0,1,1,\mathrm{make},0,\mathrm{make}\rangle & \mapsto \langle(\varepsilon,\varepsilon),\top_{\mathrm{num}}\rangle,\\ \langle 0,1,1,\mathit{left}_0,0,\mathit{left}_0\rangle & \mapsto \langle(\varepsilon,\varepsilon),\top_{\mathrm{num}}\rangle,\\ \langle 1,1,1,\mathrm{mon},0,\mathrm{mon}\rangle & \mapsto \langle((0,1).(1,2)^*,\varepsilon),x_{(0,1)}=1\rangle,\\ \langle 1,1,1,\mathit{left},0,\mathit{left}_0\rangle & \mapsto \langle((0,1),\varepsilon),x_{(0,1)}=1\rangle,\\ \langle 1,1,1,\mathit{left},0,\mathit{right}\rangle & \mapsto \left\langle \begin{array}{l}((0,1).(1,2)^*,(0,1).(1,2)^*),\\ \left\{\begin{array}{l} x_{(0,1)} = y_{(0,1)} = 1\\ x_{(1,2)} = y_{(1,2)} + 1\end{array}\right.\end{array}\right\rangle,\\ \langle 1,1,1,\mathit{right},1,\mathit{right}\rangle & \mapsto \left\langle \begin{array}{l}((0,1).(1,2)^*,(0,1).(1,2)^*),\\ \left\{\begin{array}{l} x_{(0,1)} = y_{(0,1)} = 1\\ x_{(1,2)} = y_{(1,2)}\end{array}\right.\end{array}\right\rangle,\\ \langle 1,2,1,\mathrm{make},0,\mathrm{make}\rangle & \mapsto \langle((0,1).(1,2)^*,\varepsilon),x_{(0,1)}=1\rangle,\\ \langle 1,2,1,\mathit{right},1,\mathit{right}\rangle & \mapsto \left\langle \begin{array}{l}((0,1).(1,2)^*,(0,1).(1,2)^*),\\ \left\{\begin{array}{l} x_{(0,1)} = y_{(0,1)} = 1\\ x_{(1,2)} = y_{(1,2)}\end{array}\right.\end{array}\right\rangle,\\ \langle 2,1,1,\mathrm{mon},0,\mathrm{mon}\rangle & \mapsto \langle((0,1).(1,2)^*,\varepsilon),x_{(0,1)}=1\rangle,\\ \langle 2,1,1,\mathit{left}_0,0,\mathit{left}_0\rangle & \mapsto \langle((0,1).(1,2)^*,\varepsilon),x_{(0,1)}=1\rangle,\\ \langle 2,1,1,\mathit{left},0,\mathit{left}_0\rangle & \mapsto \langle((0,1),\varepsilon),x_{(0,1)}=1\rangle,\\ \langle 2,1,1,\mathit{left},0,\mathit{right}\rangle & \mapsto \left\langle \begin{array}{l}((0,1).(1,2)^*,(0,1).(1,2)^*),\\ \left\{\begin{array}{l} x_{(0,1)} = y_{(0,1)} = 1\\ x_{(1,2)} = y_{(1,2)} + 1\end{array}\right.\end{array}\right\rangle\end{array}\right\}$$

This is a very accurate description of the communication topology of the ring. In particular, we are able to distinguish between instances of recursively defined agents and channels. □

## 6 Conclusion

We have described a parametric analysis framework for automatically inferring communication topologies of mobile systems specified in the $\pi$-calculus. We have instantiated this framework to obtain an effective analysis which is able to give finite descriptions of unbounded communication topologies that distinguish between instances of recursively defined components. To our knowledge this is the only existing analysis of mobile systems (excluding [Ven96b]) which can produce results of that level of accuracy without any strong restriction on the

base language. Previous works addressed the issue of communication analysis in CSP [CC80, Mer91] or CML [NN94, Col95a, Col95b]. In the latter papers, the analysis techniques heavily rely on CML type information and cannot be applied to more general untyped languages like the $\pi$-calculus.

In order to keep the presentation clear within a limited space, we had to make some simplifying assumptions that can be relaxed in many ways, for example by using more expressive abstract domains to denote relations between thread and channel identifiers, like cofibered domains [Ven96a, Ven99], by refining the abstract semantics to take more information into account, like the number of instances of a channel or a thread, or by considering a richer version of the $\pi$-calculus with guarded choice, matching and nested replications. Finally, in view of the encodings of classical language constructs (data structures, references, control structures) in the $\pi$-calculus, it would be interesting to study the possiblity of using a static analysis of the $\pi$-calculus as a universal analysis back-end for high-level languages.

**Acknowledgements.** I wish to thank Radhia Cousot, Patrick Cousot, Ian Mackie and the anonymous referees for useful comments on a first version of this paper.

## References

[BB92] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.

[Bou92] G. Boudol. Asynchrony and the $\pi$-calculus. Technical Report 1702, INRIA, 1992.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages*, pages 238–252, 1977.

[CC80] P. Cousot and R. Cousot. Semantic analysis of communicating sequential processes. In *Seventh International Colloquium on Automata, Languages and Programming*, volume 85 of *LNCS*, 1980.

[CC92] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of logic and computation*, 2(4):511–547, August 1992.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the Fifth Conference on Principles of Programming Languages*. ACM Press, 1978.

[Col95a] C. Colby. Analyzing the communication topology of concurrent programs. In *Symposium on Partial Evaluation and Program Manipulation*, 1995.

[Col95b] C. Colby. Determining storage properties of sequential and concurrent programs with assignment and structured data. In *Proceedings of the Second International Static Analysis Symposium*, volume 983 of *Lecture Notes in Computer Science*, pages 64–81. Springer-Verlag, 1995.

[Cou81] P. Cousot. Semantic foundations of program analysis. In S.S. Muchnick and N.D. Jones, editors, *Program Flow Analysis: Theory and Applications*, chapter 10, pages 303–342. Prentice-Hall, Inc., Englewood Cliffs, 1981.

[Deu92] A. Deutsch. A storeless model of aliasing and its abstraction using finite representations of right-regular equivalence relations. In *Proceedings of the 1992 International Conference on Computer Languages*, pages 2–13. IEEE Computer Society Press, 1992.

[DPLT96] P. Degano, C. Priami, L. Leth, and B. Thomsen. Analysis of facile programs: A case study. In *Proceedings of the Fifth LOMAPS Workshop on Analysis and Verification of Multiple-Agent Languages*, volume 1192 of *Lecture Notes in Computer Science*, pages 345–369. Springer-Verlag, 1996.

[Gra91] P. Granger. Static analysis of linear congruence equalities among variables of a program. In *TAPSOFT'91*, volume 493. Lecture Notes in Computer Science, 1991.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[Jon81] H.B.M Jonkers. Abstract storage structures. In De Bakker and Van Vliet, editors, *Algorithmic languages*, pages 321–343. IFIP, 1981.

[Kar76] M. Karr. Affine relationships among variables of a program. *Acta Informatica*, pages 133–151, 1976.

[Mer91] N. Mercouroff. An algorithm for analysing communicating processes. In *Mathematical Foundations of Programming Semantics*, volume 598 of *LNCS*, 1991.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mil91] R. Milner. The polyadic $\pi$-calculus: a tutorial. In *Proceedings of the International Summer School on Logic and Algebra of Specification*. Springer-Verlag, 1991.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation*, 100:1–77, 1992.

[NN94] H. R. Nielson and F. Nielson. Higher-order concurrent programs with finite communication topology. In 21$^{st}$ *ACM Symposium on Principles of Programming Languages*, 1994.

[Tur95] D. N. Turner. *The Polymorphic Pi-Calculus: Theory and Implementation*. PhD thesis, Edinburgh University, 1995.

[Ven96a] A. Venet. Abstract cofibered domains: Application to the alias analysis of untyped programs. In *Proceedings of the Third International Static Analysis Symposium SAS'96*, volume 1145 of *Lecture Notes in Computer Science*, pages 366–382. Springer-Verlag, 1996.

[Ven96b] A. Venet. Abstract interpretation of the $\pi$-calculus. In *Proceedings of the Fifth LOMAPS Workshop on Analysis and Verification of High-Level Concurrent Languages*, volume 1192 of *Lecture Notes in Computer Science*, pages 51–75. Springer-Verlag, 1996.

[Ven99] A. Venet. Automatic analysis of pointer aliasing for untyped programs. *Science of Computer Programming*, 1999. To appear.